Lecture 5: Model-Free Control

David Silver

All previous lectures lead to this lecture.

You drop your robot/agent into an unknown environment, we know nothing about how the environment works, how can we max the rewards? The core techniques we used in this lecture is built on lec4.

In future lectures, we will talk about how scale up.

Outline

1. Introduction
2. On-Policy Monte-Carlo Control
3. On-Policy Temporal-Difference Learning
4. Off-Policy Learning
5. Summary

Model-Free Reinforcement Learning

lec 3:

Planning (prediction, control) by DP.
Solve a known MDP.

- Lec 4 Last lecture:
  - Model-free prediction
  - *Estimate* the value function of an *unknown* MDP

Drop your agent in an unknown MDP with a given policy, how to evaluate this policy, how much rewards we can get if following the behaviors of this policy.

- Lec 5 This lecture:
  - Model-free control
  - *Optimise* the value function of an *unknown* MDP

Find v_*, q_*
We use same tools, we iterate them and find the best possible behaviors.

Uses of Model-Free Control

Why interesting? Why useful?

Some example problems that can be modelled as MDPs

- Elevator
- Parallel Parking
- Ship Steering
- Bioreactor
- Helicopter
- Aeroplane Logistics
- Robocup Soccer
- Quake
- Portfolio management
- Protein Folding
- Robot walking
- Game of Go

For most of these problems, either:

These problems are unknown to use. We don't know the environment so have to use model–free MDPs.

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

Model-free control can solve these problems

On and Off-Policy Learning

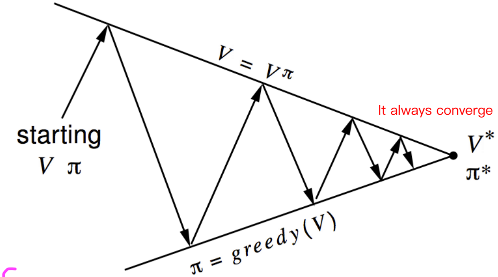
- On-policy learning

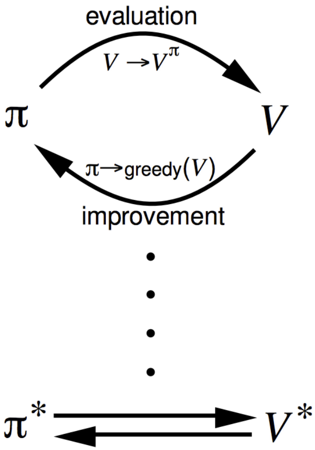
  Follow the behaviors we learn from this job.
  You get a policy, you follow that policy. While following it, you learn about that policy.

  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone's shoulder" Follow someone else's behaviors.
  - Learn about policy π from experience sampled from μ

    The robot/agent can learn not only from itself's experience but also others'. Other can be other robot/agent or even human demonstrations.

Generalised Policy Iteration (Refresher)





Repeat

Firstly

Policy evaluation Estimate v_π

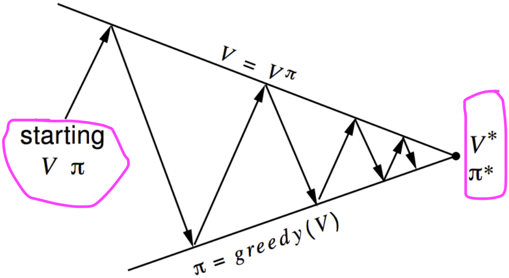
e.g. Iterative policy evaluation

Then

Policy improvement Generate $\pi' \geq \pi$

e.g. Greedy policy improvement

Generalised Policy Iteration With Monte-Carlo Evaluation



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Take mean value

Policy improvement Greedy policy improvement?

Will this MC + Greedy combination work? It has 2 issues:

1. Evaluation Step: If we use status value function V, we need look ahead one step to use value func of next state while need to know the action to be taken but this makes it not model–free (because asking for know the action). Solve: Use Action value function instead (see next slide).

One more small issue: It will be slow. It needs lots of efforts to do so. This can be improved by TD later.

2. Improvement Step: If you always greedy, you will not explore the whole state space. So there might be some potential you never see.

Model-Free Policy Iteration Using Action-Value Function

To replace State–Value function V

- Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

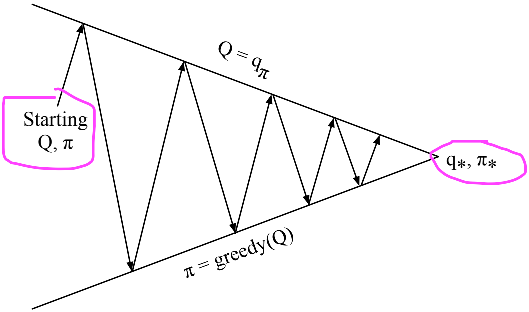
Which makes it not model free anymore

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s, a)$$

Q tells us how good to take each action
Then we find the Max. No need model here.

Generalised Policy Iteration with Action-Value Function



Solution to Issue on Policy evaluation step:

Replace V=v_pi

Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

How to solve the issue on Improvement step?

Example of Greedy Action Selection

"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

So if greedy you get stuck with "right door". You never know what is the reward of "2nd/ 3rd/4th .. time choosing left door". (This is actually the drawbacks of greedy algorithm.)

- There are two doors in front of you.

Time

1
- You open the left door and get reward 0
  $V(left) = 0$

2
- You open the right door and get reward +1
  $V(right) = +1$ MC: choose right bc Mean=1
  Greedy: choose right bc CurrentReward=1

3
- You open the right door and get reward +3
  $V(right) = +2$ MC: choose right bc Mean=1.5
  Greedy: choose right bc CurrentReward=2

4
- You open the right door and get reward +2
  $V(right) = +2$ MC: choose right bc Mean=5/3
  Greedy: choose right bc CurrentReward=2

$\vdots$

- Are you sure you've chosen the best door?

Epsilon

ϵ-Greedy Exploration

How to guarantee you visit all states or all actions?

Guarantee visit all

- Simplest idea for ensuring continual exploration But work well and efficiently
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

a=greedy(Q) if 1–epsilon
a=random a if epsilon

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a\in\mathcal{A}}{\operatorname{argmax}}\ Q(s,a) \quad \text{greedy} \\ \epsilon/m & \text{otherwise} \quad \text{random} \end{cases}$$

Fancy way to show same thing:
π (a | s) = P [A_t= a | S_t= s]

e.g.: if we have 10 action options (m=10), named a1, a2 .. a10, and a10 is best option, and epsilon=30% for random.
Then possibilities for all 10 action options:
P(a1)=P(a2)=P(a3)=...=P(a9)=0.03. They only will be chosen in 30% case.
P(a10)=0.73. It will be chosen in 30% case and 70% case.

ϵ-Greedy Policy Improvement

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

action value func q_π (s, a) = E_π [G_t | S_t = s, A_t = a]

Take one step ahead following new policy pi’

$$
\begin{aligned}
\underline{q_\pi(s, \pi'(s))} &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s,a) \\
&= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \underline{\max_{a \in \mathcal{A}} q_\pi(s,a)}
\end{aligned}
$$

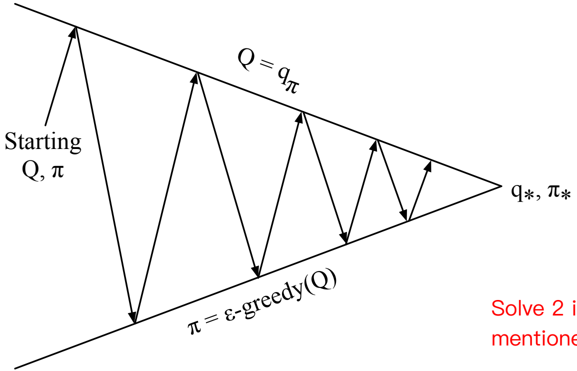
bc max >= mean

$$
\begin{aligned}
&\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \underline{\sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1-\epsilon} q_\pi(s,a)} \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) = \underline{v_\pi(s)}
\end{aligned}
$$

Better than taking one step ahead following old policy pi

Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

Monte-Carlo Policy Iteration



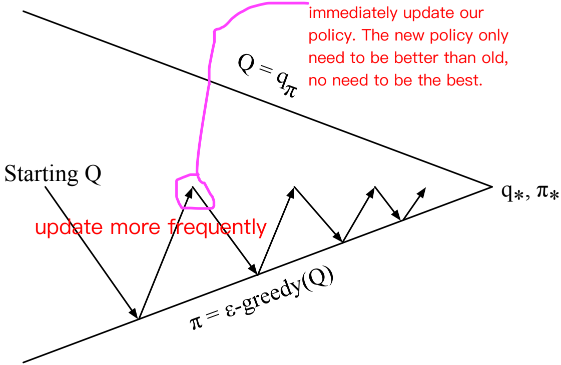
Solve 2 issues we mentioned before.

Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$ Replace V=v_pi

Policy improvement ϵ-greedy policy improvement Replace greedy.

Monte-Carlo Control

more efficient



After one episode, we immediately update our policy. The new policy only need to be better than old, no need to be the best.

update more frequently

Idea is always act greedy to wrt the most freshest most recent estimated action–value function. After one episode, you can update the value function slightly better, instead of using old estimated action–value function to generate action, use your new updated estimated action–value function to generate behavior.

Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

How can we guarantee we find the pi_*? We should balance two things. 1 we keep exploring and don't exclude anything which can make it better. 2 asymptotically we get to a policy we're not exploring at all anymore bc the best policy don't include the random behavior.

GLIE

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,

  Every action from one state will be tried

$$\lim_{k\rightarrow\infty} N_k(s,a) = \infty$$

- The policy converges on a greedy policy,

  It needs to meet the bellman optimality equation which has a max.

$$\lim_{k\rightarrow\infty} \pi_k(a|s) = \mathbf{1}(a = \underset{a'\in\mathcal{A}}{\operatorname{argmax}}\ Q_k(s,a'))$$

- For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE Monte-Carlo Control

- Sample kth episode using π: $\{S_1, A_1, R_2, ..., S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

counter: $$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

Update the mean: $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left(G_t - Q(S_t, A_t)\right)$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$

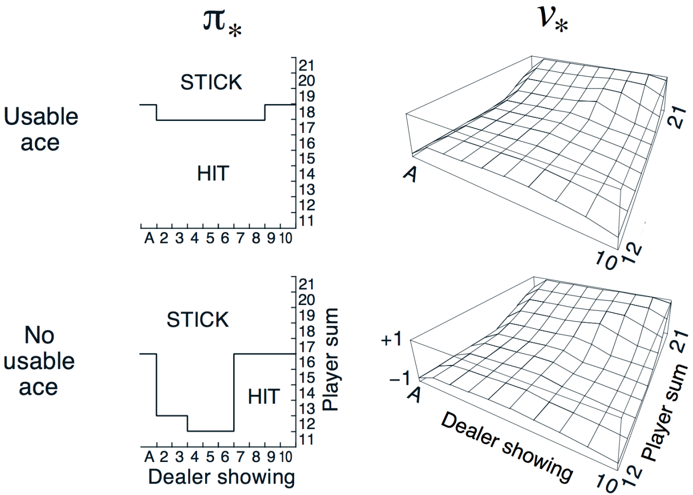
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

GLIE Monte-Carlo control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

Back to the Blackjack Example

Monte-Carlo Control in Blackjack

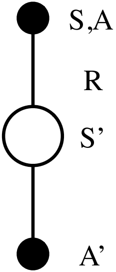


MC vs. TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy policy improvement
  - Update every time-step from every episode

Updating Action-Value Functions with Sarsa

I’m in state S, I’m considering if I actually take an action, how many rewards I will get, and the value of next action I would take. This is estimated value of that policy and used to update the value of state–action pair I started in.



State–action pair. Start with state S, choosing an action,

sample from environment to get reward R,

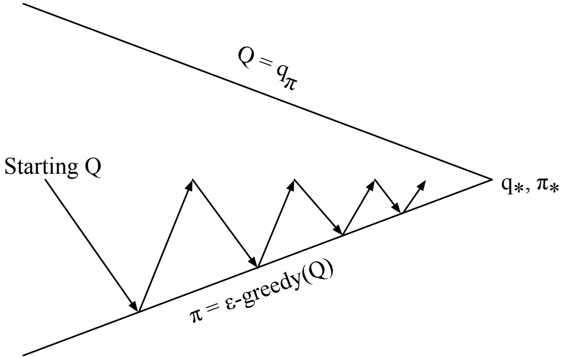
end with new state S’. Then sample our own policy at next state S’.

$$Q(S,A) \leftarrow Q(S,A) + \alpha \overbrace{\left(\overbrace{R + \gamma Q(S',A')}^{\text{TD target}} - Q(S,A)\right)}^{\text{TD error}}$$

TD target: Immediate reward + discounted value of next state

– q value where we started

On-Policy Control With Sarsa



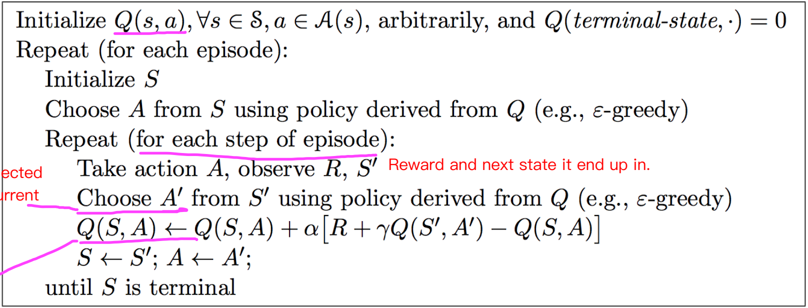
Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Sarsa Algorithm for On-Policy Control

A lookup table



```
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Reward and next state it end up in.

A’ is selected using current policy.

Have a new policy now

Convergence of Sarsa

Theorem

Sarsa converges to the optimal action-value function, As GIIE–MC
$Q(s, a) \to q_*(s, a)$, *under the following conditions:*

- *GLIE sequence of policies* $\pi_t(a|s)$
- *Robbins-Monro sequence of step-sizes* α_t

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

Step size is sufficiently large so you can move your q value as far as you want.

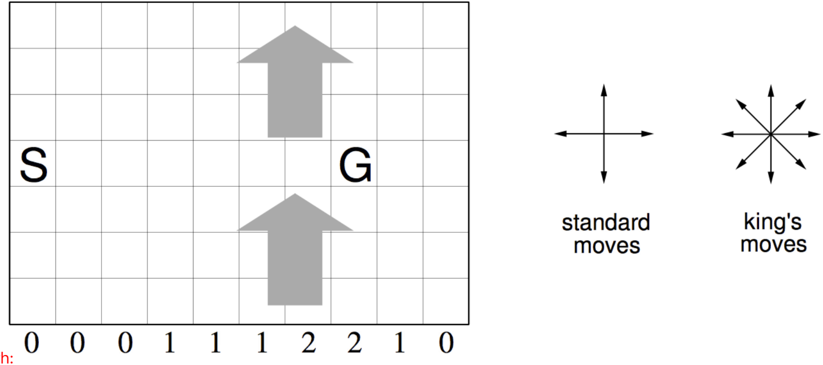
$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Eventually the changes of your q value becomes smaller and smaller and to 0.

Empirical result: we often don’t worry about these 2 so salsa typically works.

Windy Gridworld Example

Move from start cell S to goal cell G. Use king's moves. Each move, the wind will move us up by wind strength pieces of cells.

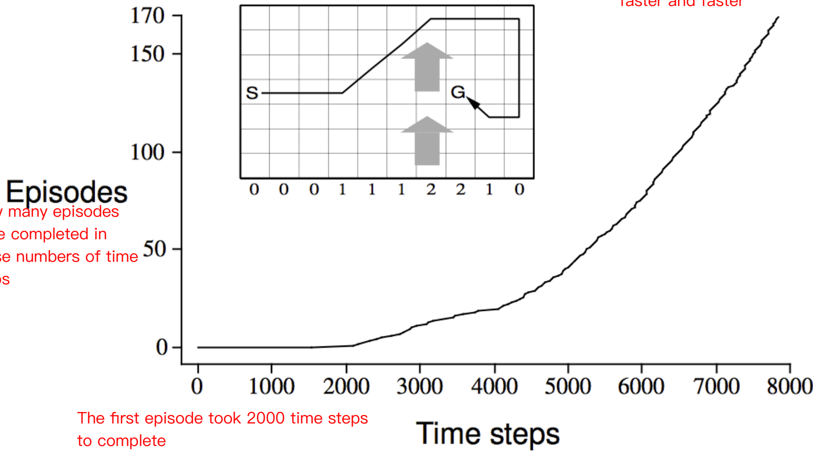


Wind strength:

- Reward = -1 per time-step until reaching goal
- Undiscounted

Sarsa on the Windy Gridworld

The slope is showing it learns faster and faster



How many episodes were completed in those numbers of time steps

The first episode took 2000 time steps to complete

n-Step Sarsa

- Consider the following n-step returns for $n = 1, 2, \infty$:

$$\begin{array}{lll} n = 1 & (Sarsa) & q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\ n = 2 & & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\ \vdots & & \vdots \\ n = \infty & (MC) & q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T \end{array}$$

1 step ahead

2 steps ahead

No bootstrap

- Define the n-step Q-return

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} \boxed{+} \gamma^n Q(S_{t+n})$$

N steps immediate rewards + estimated rewards for all remaining steps until end of the episode

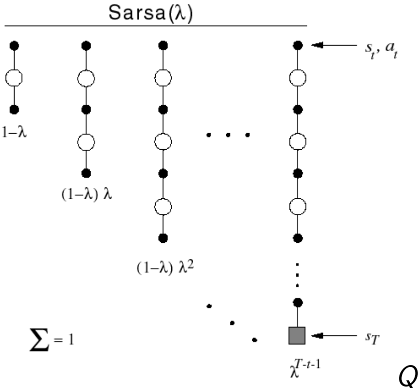
- n-step Sarsa updates $Q(s, a)$ towards the n-step Q-return

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(q_t^{(n)} - Q(S_t, A_t) \right)$$

Update our estimated value of taking action A_t at state S_t a little bit in the direction of our n steps target

Forward View Sarsa(λ)

A spectrum between Monte Carlo and TD(0)



- The q^λ *return* combines all *n*-step Q-returns $q_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}q_t^{(n)}$$

average all n returns

- Forward-view Sarsa(λ)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(q_t^\lambda - Q(S_t, A_t)\right)$$

Problem: this isn't online algorithm. We cannot update our Q value immediately and improve our policy. We have to wait until the end of our episode to calculate the q^lambda.
We want to run thing online and get the freshest possible updates immediately and improve our policy at every single step.

Backward View Sarsa(λ)

Solution to problem in previous slide: Build the equivalence: eligibility trace.

- Just like TD(λ), we use eligibility traces in an online algorithm
- But Sarsa(λ) has one eligibility trace for each state-action pair

$$E_0(s,a) = 0$$

A table to record who is responsible (credited or blamed) for the received (positive or negative) rewards.

$$E_t(s,a) = \underbrace{\gamma\lambda E_{t-1}(s,a)}_{\text{decay}} + \underbrace{\mathbf{1}(S_t = s, A_t = a)}_{\text{Bump up eligibility}}$$

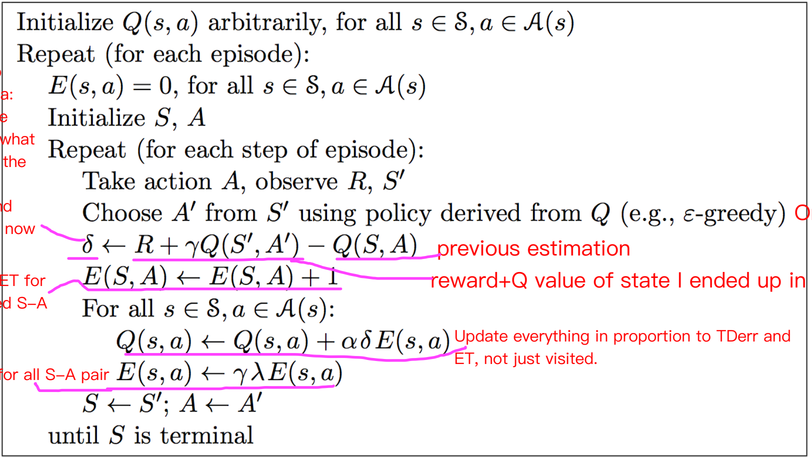
- $\underline{Q(s,a) \text{ is updated for every state } s \text{ and action } a}$
- $\underline{\text{In proportion to TD-error } \delta_t \text{ and eligibility trace}}$ $\underline{E_t(s,a)}$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$\underline{Q(s,a) \leftarrow Q(s,a) + \alpha \delta_t E_t(s,a)}$$

Issue: table lookup is naive and cannot solve large scale problems. Solution: next lecture, function approximation.

Sarsa(λ) Algorithm



```
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
    $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
    Initialize $S$, $A$
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $\delta \leftarrow R + \gamma Q(S',A') - Q(S,A)$
        $E(S,A) \leftarrow E(S,A) + 1$
        For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
            $E(s,a) \leftarrow \gamma \lambda E(s,a)$
        $S \leftarrow S'$; $A \leftarrow A'$
    until $S$ is terminal
```

TD 1 step error delta: Difference between what I thought the value is before and what it is now

On policy

previous estimation

reward+Q value of state I ended up in

Increase ET for just visited S–A pair

Update everything in proportion to TDerr and ET, not just visited.

Decay ET for all S–A pair

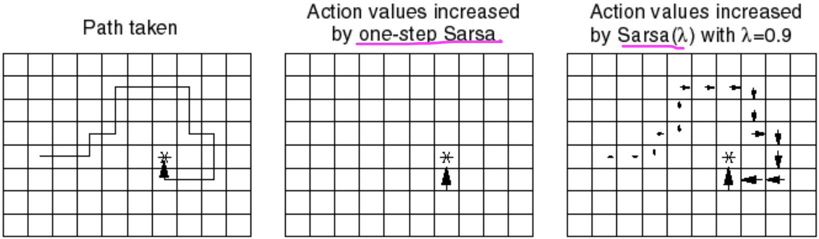
Sarsa(λ) Gridworld Example

In this example the forward view and backward view are equivalent.
But the computation of backward view is much nicer, it's online, just keep one step in memory, no need waiting for end of episode.

All initial value is 0.

Arrow size means how big the Q value of S–A pair is.

Lambda value decides how quickly and how far that information should propagate back through your trajectory.


Path taken
Action values increased by one-step Sarsa
Action values increased by Sarsa(λ) with λ=0.9

All path cell values are still 0. Only the last cell has been changed to 1 (the reward).
You only propagate your information back by one step per episode. So only the last cell get updated. If want all path cell values get update you need lots of episode.

You built your ET all along your trajectory. Each cell (S–A pair) you visited has ET. The older ones decay more the one more recent decay less. When you see the reward of 1 at the end, you increase all those cell (pair) in proportion to TD err and ET. All of them get updated to the direction of what happened. The information flow backwards in one episode.

Off-Policy Learning

All before are on-policy learning. The policy I follow is the one I learn about.

Not same policy

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$
- While following behaviour policy $\mu(a|s)$

The behavior here in /mu is not from policy /pi.

$$\{S_1, A_1, R_2, ..., S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

Not just supervised learning to copy what human did, but learn to solve the MDP from watching their experiences.

Importance Sampling

One of 2 mechanisms for off-policy learning

- Estimate the expectation of a different distribution

Possibilities

Rewards you get

$$\begin{aligned}\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

Q(X) : other distribution

Expectation over other distribution

Correct the changes between your distributions

Importance Sampling for Off-Policy Monte-Carlo

Every single step, there are some actions I took according to the policy /mu I follow; there are some probability I would take action from the policy I am learning about /pi. We multiply those ratios together, and get a much smaller probability but the return I saw under my behavior policy /mu, actually matched giving us information about what would happen if I follow /pi.

- Use returns generated from μ to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

Ratios

Along entire trajectory

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \dots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value towards *corrected* return

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{\pi/\mu} - V(S_t) \right)$$

- Cannot use if μ is zero when π is non-zero
- Importance sampling can dramatically increase variance

You can use this idea but IS for Monte Carlo learning is very variance and in practice it's useless.
So you have to use TD learning for off–policy which is imperative to bootstrap.

Importance Sampling for Off-Policy TD

You only need to importance sample over one step now bc we bootstrap after one step.

- Use TD targets generated from μ to evaluate π
- Weight TD target $R + \gamma V(S')$ by importance sampling
  1 step
- Only need a single importance sampling correction

Just correct our distribution over one step

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\underbrace{\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \left(R_{t+1} + \gamma V(S_{t+1})\right)}_{\text{TD target}} - V(S_t) \right)$$

- Much lower variance than Monte-Carlo importance sampling
  It still increase the variance by importance sampling.
- Policies only need to be similar over a single step

Q-Learning

2nd of 2 mechanisms for off-policy learning

The idea works best with off–policy learning is Q–learning. This is TD(0)/Sarsa(0) now.

Make use of action values Q

- We now consider off-policy learning of action-values $Q(s, a)$
- No importance sampling is required

We really take

Idea:

- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$ Target policy

We image what would be if we take, in target policy.

- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

Compare:

Sarsa: Q (S, A) ← Q (S, A) + α (R + γ Q (S ′ , A ′) − Q (S, A))

Sarsa Forward: Q (S_t , A_t) ← Q (S_t , A_t) + α (q_t^λ − Q (S t , A t))

Sarsa Backward:

δ_t = R_t+1 + γ * Q (S_t+1 , A_t+1) − Q (S_t , A_t)
Q (s, a) ← Q (s, a) + α * δ_t * E_t (s, a)

The reward we get by take action A_t at S_t

The discounted value of the next state S_t+1 of my alternative action on my target policy.

Off-Policy Control with Q-Learning

A special case of this is well–known Q–learning.

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

We try to learn about greedy behavior while we following exploratory behavior.

$$A' = \quad \pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')$$

Both the behavior and target policy can improve. We add improvement steps to both of them.

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

TD target

$$R_{t+1} + \gamma Q(S_{t+1}, \underline{A'})$$

Target from last slide

$$\overset{=}{}$$

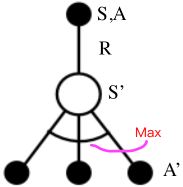
$$= R_{t+1} + \gamma Q(S_{t+1}, \underline{\underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')})$$

$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

It updates a little bit in the direction of max Q value you can get

Q-Learning Control Algorithm

Can thinks as Sarsa–MAX



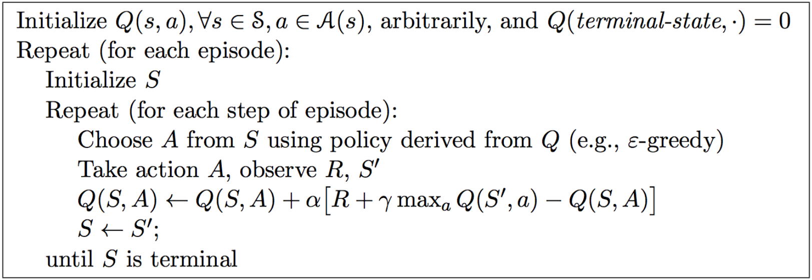
$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S',a') - Q(S,A) \right)$$

Update our Q value a little bit in the direction of best possible next Q value we could have after one step. This is also bellman optimality equation.

Theorem

Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow q_*(s,a)$

Q-Learning Algorithm for Off-Policy Control

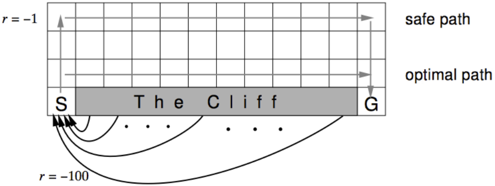


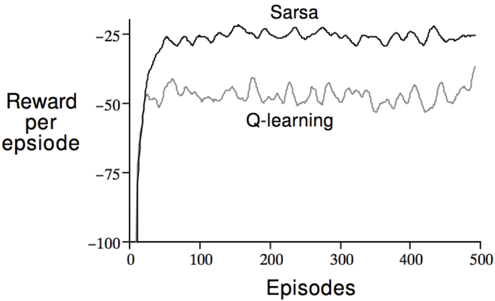
```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

Q-Learning Demo

Q-Learning Demo

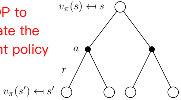
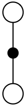
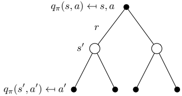
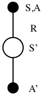
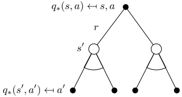
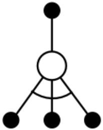
Cliff Walking Example





Relationship Between DP and TD

Sample based algorithms

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$ State–value function	Use DP to evaluate the current policy  $v_\pi(s) \leftarrow\!\shortmid s$ a r $v_\pi(s') \leftarrow\!\shortmid s'$ Iterative Policy Evaluation	 TD learning take one sample of left cell area (DP). Targets of TD learning are samples of DP. TD Learning
Bellman Expectation Equation for $q_\pi(s,a)$ Action–value function	 $q_\pi(s,a) \leftarrow\!\shortmid s,a$ r s' $q_\pi(s',a') \leftarrow\!\shortmid a'$ Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s,a)$	 $q_*(s,a) \leftarrow\!\shortmid s,a$ r s' $q_*(s',a') \leftarrow\!\shortmid a'$ Q-Value Iteration	 Q-Learning

Relationship Between DP and TD (2)

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	TD Learning $V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	Sarsa $Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S',A')$
Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	Q-Learning $Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

Questions?